Программирование

Python

Лекция 12

Кафедра ИВТ и ПМ

2018

План

Прошлые темы

Функции

- Параметр (аргумент) по умолчанию
- Процедурное программирование
- Программирование методом сверху вниз
- Механизм вызова функций

Рекурсия

- Рекурсия и итерация

Outline

- Что такое область видимости?
- Что такое глобальная переменная?
- Что такое локальная переменная?
- Как описывается функция?

- Что такое область видимости?
- Что такое глобальная переменная?
- Что такое локальная переменная?
- Как описывается функция?
- Как в Python определить тип?

- Что такое область видимости?
- Что такое глобальная переменная?
- Что такое локальная переменная?
- Как описывается функция?
- Как в Python определить тип?

```
type( переменная или выражение ) -> тип
```

- Какого типа будет возвращаемое значение функции?

```
def foo():
    ...
    x = 10
    y = True
    z = "To be, or not to be"
    return x, y, z


type( foo() ) # -> ?
```

- Какого типа будет возвращаемое значение функции?

```
def foo():
    ...
    x = 10
    y = True
    z = "To be, or not to be"
    return x, y, z


type( foo() ) # -> ?
```

tuple (кортеж)

- Какого типа будет возвращаемое значение функции?

```
def bar():
    ...


type( bar() ) # -> ?
```

- Какого типа будет возвращаемое значение функции?

```
def bar():
    ...


type( bar() ) # -> ?
```

NoneType

Пример 1

Требуется вычислять следующее выражение [1]

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Логично представить его как функцию.

Какие формальные параметры будет иметь функция?
Как должна выглядеть функция?

[1] функция плотности нормального распределения случайной величины

Пример 1

```
def norm_pdf (x, mu, sigma):
    return 1 / (sigma * sqrt(2*pi) )  * \
            exp( - (x-mu)**2 / 2/sigma**2 )


norm_pdf(0, 0, 1)
norm_pdf(0.5, 0, 1)
norm_pdf(1, 0, 1)
```

Пример 1

Как построить график функции f(x)?

Пример 1

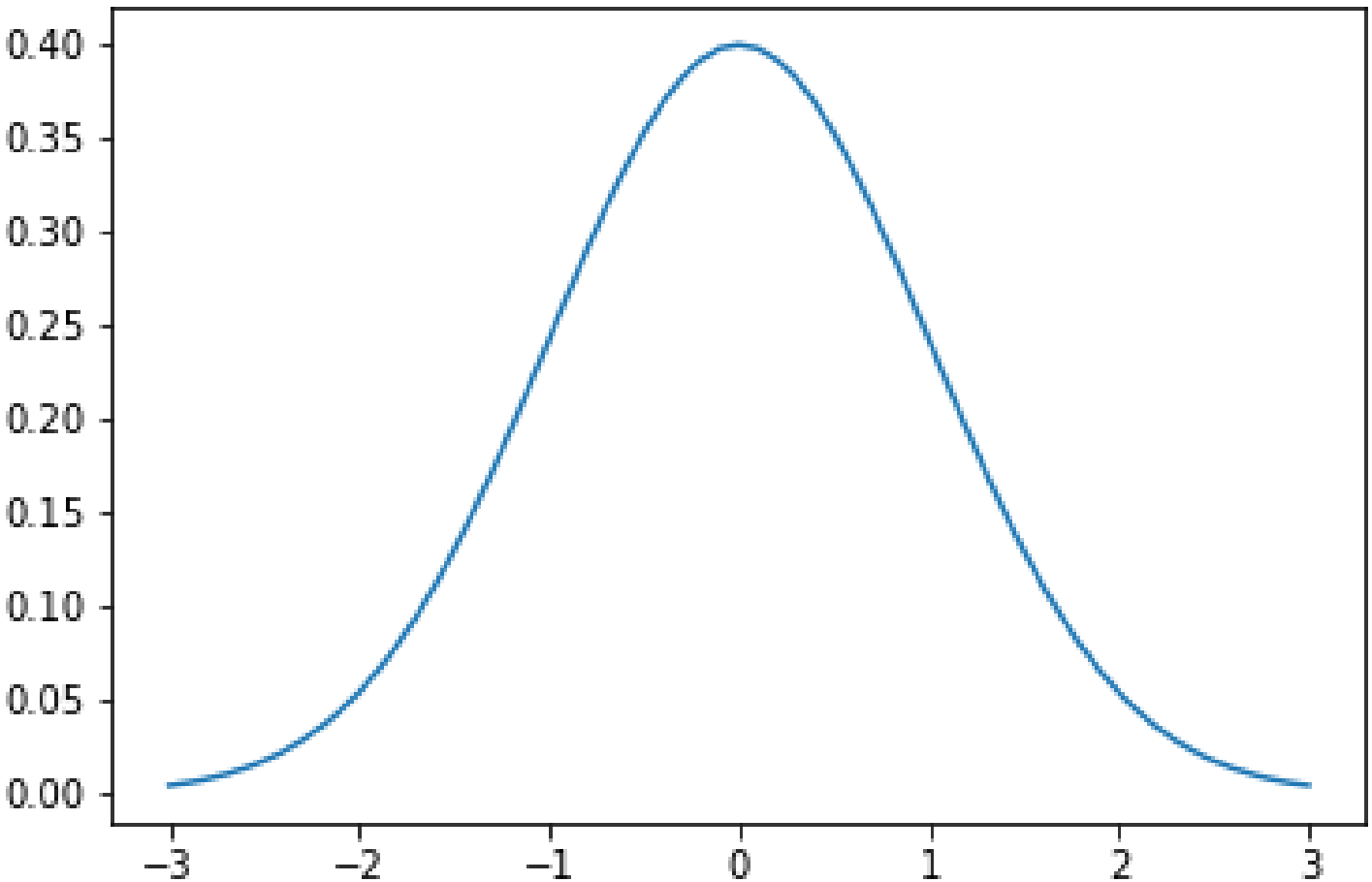
Как построить график функции f(x)?

```
from math import *
from matplotlib.pyplot import *

def norm_pdf (x, mu, sigma):
    return 1 / (sigma * sqrt(2*pi) )  * \
            exp( - (x-mu)**2 / 2/sigma**2 )

X = [x/10 for x in range(-30, 31)]
Y = [norm_pdf(x, 0, 1) for x in X]
plot(X,Y)
show()
```

Пример 1



Пример 2

Задача: сложить два списка из целых числе поэлементно.
Списки имеют одинаковые длины.

```
def sum_lists(L1, L2):
    for i in range(len(L1)):
        print( L1[i] + L2[i], end=" ")
```

```
A = [1,2,3]
B = [4,5,6]

sum_lists(A,B) # 5 7 9
```

В чём основная проблема программы?

Пример 2

Проблема:
в функции смешаны программный интерфейс и интерфейс пользователя. Входные параметры задаются через программный интерфейс, а выходные видны только в интерфейсе пользователя.

Таким образом невозможно получить результат вызова функции, например записать его в переменную, хотя это может понадобится. Например когда нужно поэлементно сложить три списка.

Как должна быть организована программа:

Пример 2

Как должна быть организована программа[2]:

```
def sum_lists(L1, L2):
    L = []
    for i in range(len(L1)):
        L += [ L1[i] + L2[i] ]
    return L

def print_list(L):
    for e in L:
        print(e, end = ' ')

A = [1,2,3]
B = [4,5,6]
C = sum_lists(A,B)
print_list(C)   # 5 7 9
```

[2]Программу можно улучшить использовав генератор списков в функции sum_lists

Outline

Outline

Следующая функция часто вызывается с одинаковым значением параметров mu = 0, sigma = 1.

```
def norm_pdf (x, mu, sigma):
    return 1 / (sigma * sqrt(2*pi) )  * \
            exp( - (x-mu)**2 / 2/sigma**2 )
```

Однако от этих формальных параметров нельзя отказаться вовсе, сделав mu и sigma локальными переменными функции и задав им значения 0 и 1 соответственно.

Параметр (аргумент) по умолчанию

В языках программирования есть способ задавать значения формальным параметрам во время определения функции.

Такие параметры называются **параметрами по умолчанию**.

```
def norm_pdf (x, mu = 0, sigma = 1):
    return 1 / (sigma * sqrt(2*pi) )  * \
        exp( - (x-mu)**2 / 2/sigma**2 )
```

Параметр (аргумент) по умолчанию

Формальному параметру по умолчанию не обязательно должен соответствовать фактический параметр, потому что значение формального параметра уже задано.

Но если фактический параметр всё же использован, то будет взято именно его значение.

```
def norm_pdf (x, mu = 0, sigma = 1):
    return 1 / (sigma * sqrt(2*pi) )  * \
        exp( - (x-mu)**2 / 2/sigma**2 )

norm_pdf(0.5)          # mu = 0, sigma = 1
norm_pdf(0.5, 7)       # mu = 7, sigma = 1
norm_pdf(0.5, 7, 3)    # mu = 7, sigma = 3

# следующие вызовы равнозначны
norm_pdf(0.5)          # mu = 0, sigma = 1
norm_pdf(0.5, 0)       # mu = 0, sigma = 1
norm_pdf(0.5, 0, 1)    # mu = 0, sigma = 1
```

Параметр (аргумент) по умолчанию

- Формальному параметру со значением по умолчанию не обязательно должен соответствовать фактический параметр
- Аргументы по умолчанию должны быть определены в заголовке функции
- Аргументов по умолчанию может быть сколько угодно. Например параметрами со значением по умолчанию могут быть все формальные параметры функции.
- Параметры по умолчанию стоит использовать когда нужна возможность передать параметр в функцию, но это делается редко.

Outline

Процедурное программирование

Процедурное программирование — программирование на императивном языке, при котором последовательно выполняемые операторы можно собрать в подпрограммы, то есть более крупные целостные единицы кода.

Процедурное программирование

Преимущества

- **Алгоритмическая декомпозиция.** Задача разбивается на подзадачи, каждая из которых решается относительно независим от от других.
  - Программа стоящая из вызовов подпрограмм легче понимания
  - В такой программе проще локализовать и исправить ошибку
- **Повторное использование кода.** Однажды написанную подпрограмму можно использовать многократно
- **Сокрытие сложности.** Для использования подпрограммы достаточно минимальных знаний об её устройстве. Не нужно знать всех деталей реализации.

Outline

Программирование "сверху вниз"

Сначала пишется текст основной программы, в котором, вместо каждого связного логического фрагмента текста, вставляется вызов подпрограммы, которая будет выполнять этот фрагмент.

Вместо настоящих, работающих подпрограмм, в программу вставляются фиктивные части — заглушки, которые, говоря упрощенно, ничего не делают.

Программирование "сверху вниз"

После того, как программист убедится, что подпрограммы вызываются в правильной последовательности (то есть общая структура программы верна), подпрограммы-заглушки последовательно заменяются на реально работающие, причём разработка каждой подпрограммы ведётся тем же методом, что и основной программы.

Разработка заканчивается тогда, когда не останется ни одной заглушки.

Программирование "сверху вниз"

Такая последовательность гарантирует, что на каждом этапе разработки программист одновременно имеет дело с обозримым и понятным ему множеством фрагментов, и может быть уверен, что общая структура всех более высоких уровней программы верна.

Пример

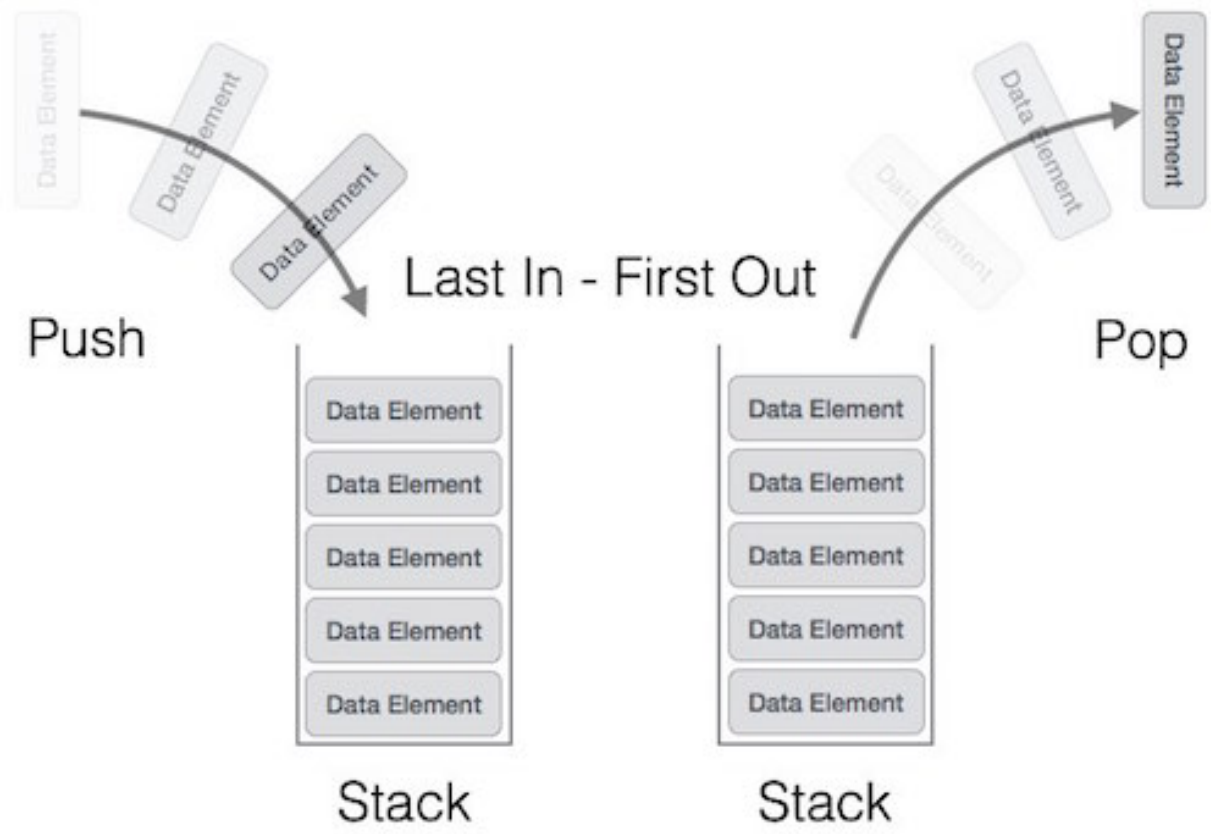
Требуется создать программу для работы с БД состоящей из одной таблицы.

```
def load_database(filaneme):
    """Загражает данные из БД в список.
    return список из записей
    """
    return [tuple()]
def print_record(d, m):
    """Печатает запись (в строку длиной не более m символов)"""
    pass
def print_database(data, n, m = 80):
    """Выводит на экран n записей из БД
    """
    for d in data:
        print_record(d, m)
N = 20 # число записей БД выводимых на экран
Data = []
filename = "mydb"  # написать функцию
if filename:
    Data = load_database(filename)
    print_database(Data, N)
...
```

Outline

Абстрактный тип данных. Пример - стек

Стек - список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»)



Вызов функций. Стек вызовов

Стек вызовов (call stack) — в теории вычислительных систем,стек, хранящий информацию для возврата управления из подпрограмм (процедур) в программу (или подпрограмму, при вложенных или рекурсивных вызовах).

Помимо адресов возврата в стеке вызовов могут хранится локальные переменные функций, параметры и другая информация. Такой блок информации для отдельной функции называется **стековым кадром** (фреймом).

Вызов функций

При вызове подпрограммы или возникновении прерывания, в стек вызовов заносится адрес возврата — адрес в памяти следующей инструкции приостановленной программы и управление передается подпрограмме

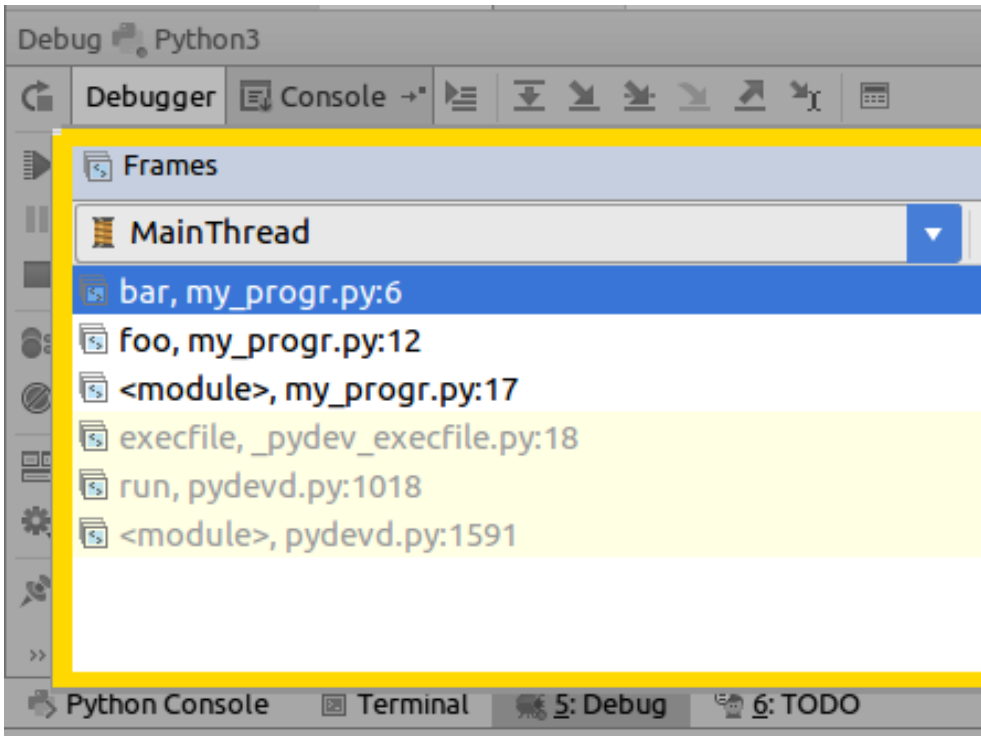
При последующем вложенном или рекурсивном вызове, прерывании подпрограммы или обработчика прерывания, в стек заносится очередной адрес возврата и т. д.

При возврате из подпрограммы или обработчика прерывания, адрес возврата снимается со стека и управление передается на следующую инструкцию приостановленной (под-)программы

Стек вызовов

Просмотр стека вызовов внутри отладчика PyCharm.
Выполнение программы остановлено на строке 6.

```
1       def baz():
2           print("baz")
3
4
5       def bar():
6           print("bar")
7           baz()
8
9
10      def foo():
11          print("foo")
12          bar()
13
14
15      foo()
```



PyCharm вместо адресов в стеке вызовов (справа) используются номера строк.

Информация о стеке вызовов

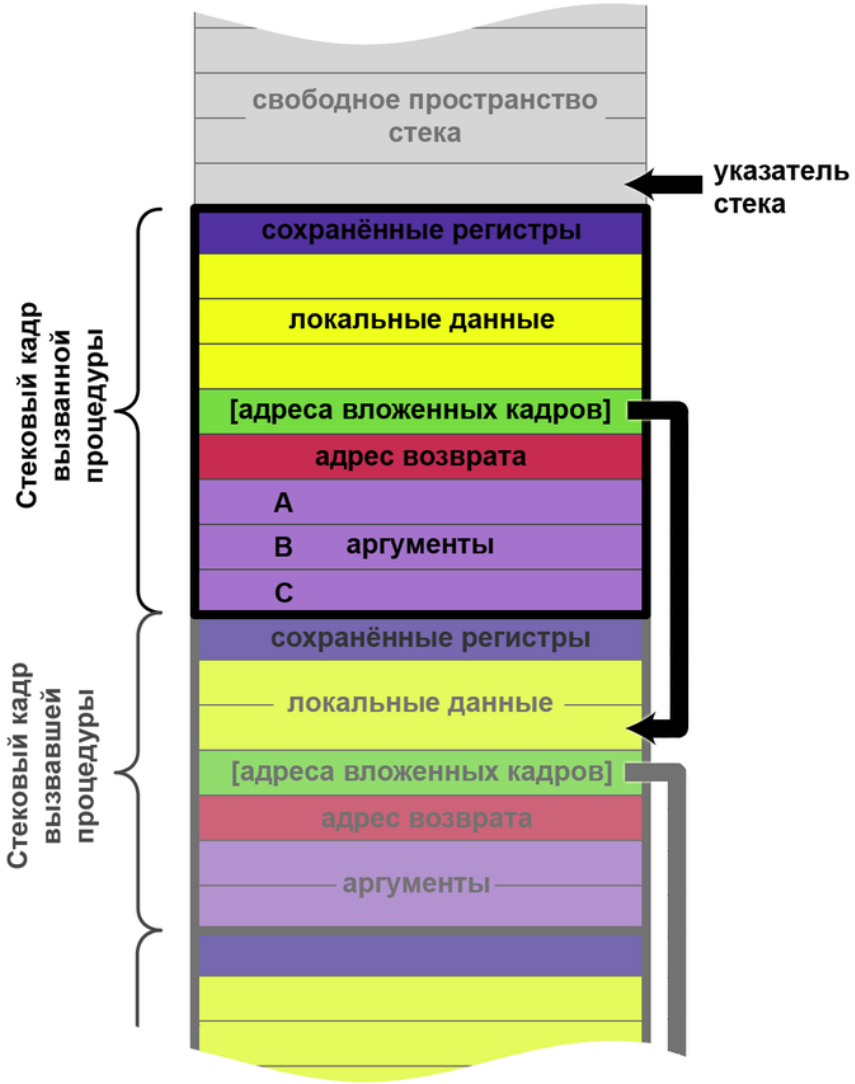
```
import inspect

 # возвращает список кадров стека вызовов
inspect.stack()
```

Кадр:

```
FrameInfo(frame=<frame object at 0x7f403a416c48>,
filename='/home/user/PycharmProjects/my_project/my_progr.py
lineno=5, function='baz',
code_context=['    print(inspect.stack())\n'], index=0)
```

Стек вызовов



В стек вызовов кроме адреса возврата могут быть помещены локальные переменные и аргументы функции.

Outline

Рекурсия

Рекурсия - определение, описание какого-либо объекта или процесса внутри самого этого объекта или процесса, то есть ситуация, когда объект является частью самого себя.

Рекурсия - определение некоторого понятия через самое себя.

Рекурсивный алгоритм – это алгоритм, в описании которого прямо или косвенно содержится обращение к самому себе.

Рекурсия

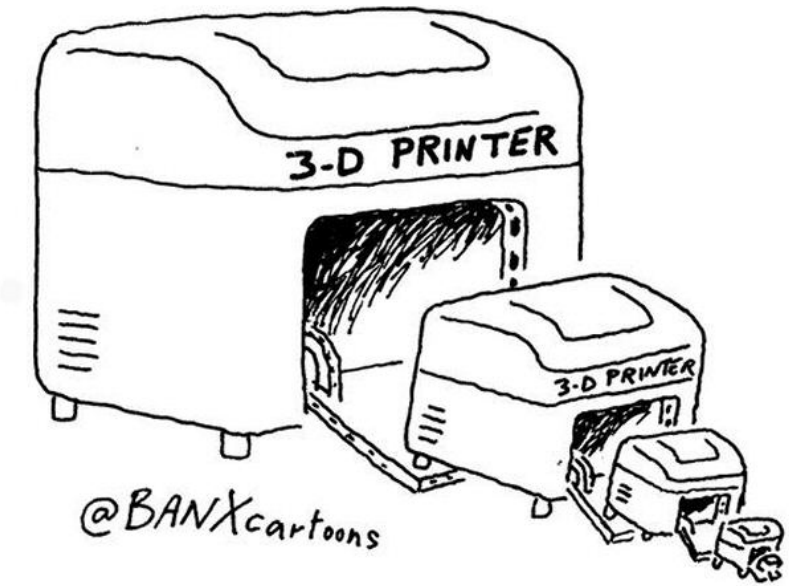
Сепульки — важный элемент цивилизации ардритов с планеты Энтеропия. См. Сепулькарии.

Сепулькарии — устройства для сепуления.

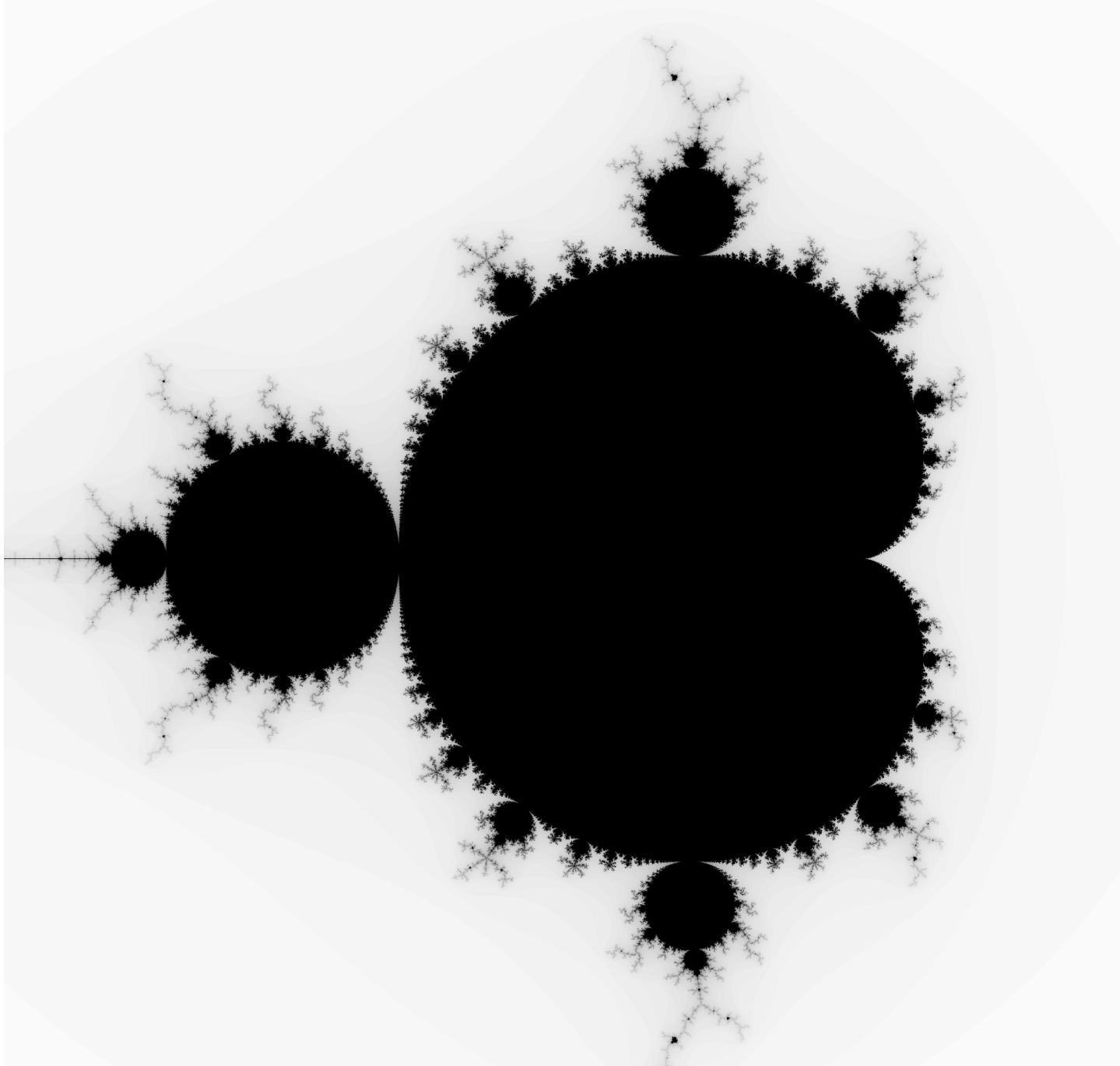
Сепуление — занятие ардритов с планеты Энтеропия. См. Сепульки[3].

[3] Станислав Лем «Звёздные дневники Ийона Тихого

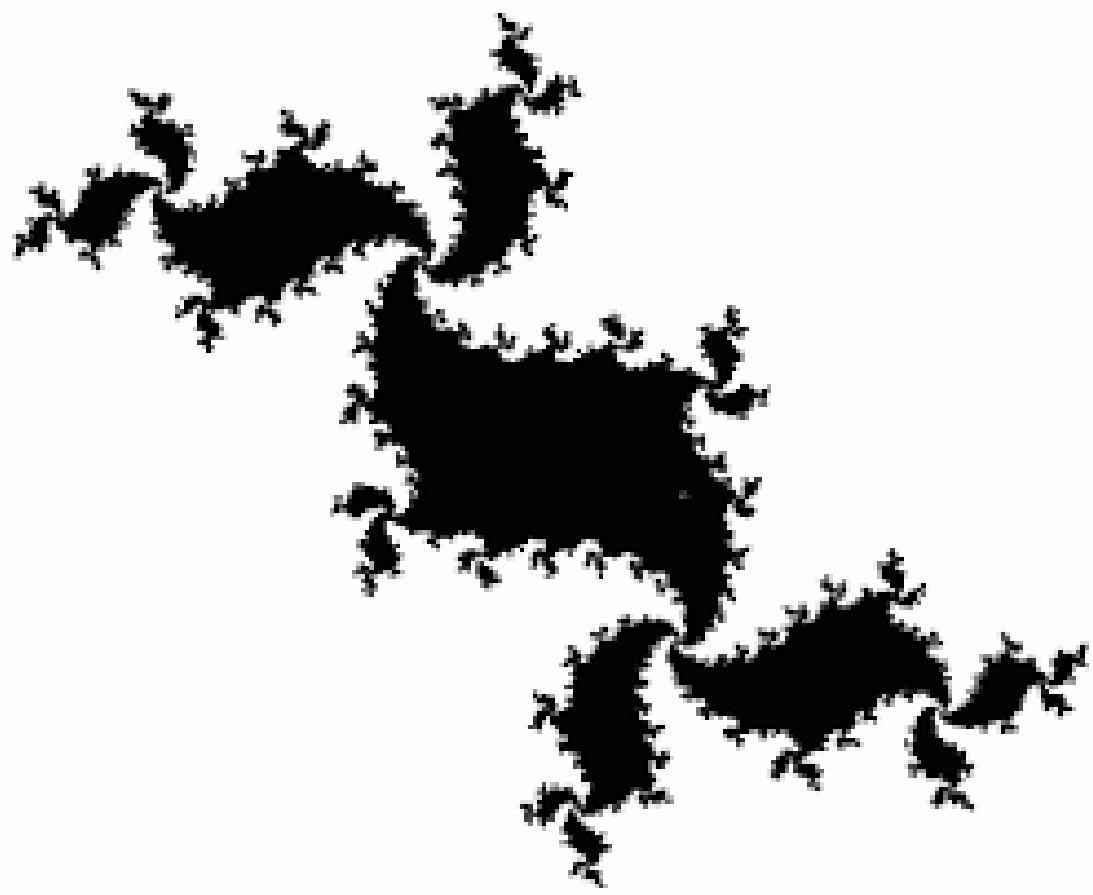
Рекурсия



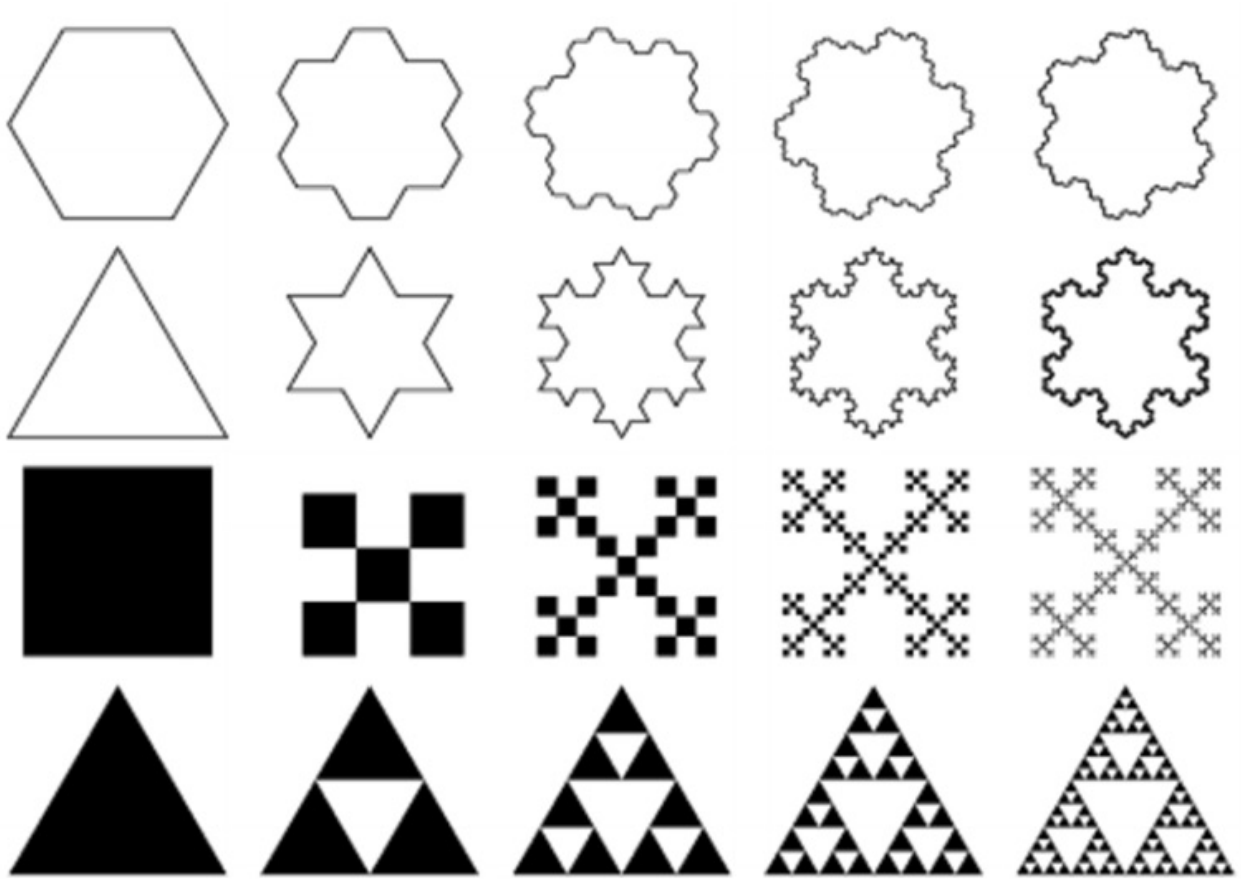
Рекурсия



Рекурсия



Рекурсия



Рекурсия

Пример рекурсивной функции: факториал

$$n! = \begin{cases} n \cdot (n-1)!, & n > 0 \\ 1, & n = 0 \end{cases}$$

Рекурсия

Пример рекурсивной функции: функция задающая последовательность Фибоначчи

1 1 2 3 5 8 13 ...

$$F = \begin{cases} F(0) = 1; \\ F(1) = 1; \\ F(n) = F(n-1) + F(n-2), \quad n > 1. \end{cases}$$

Рекурсия

база — аргументы, для которых значения функции определены (тривиальные случаи).

шаг рекурсии — способ сведения задачи к более простым

Составление рекурсивного алгоритма

- параметризация
  Какие параметры будет иметь функция?
- выделение базы
  В каком случае результат функции очевиден и не требует вычислений?
- декомпозиция
  Как разбить задачу на подзадачу?

Рекурсия

Структурно рекурсивная функция на верхнем уровне всегда представляет собой команду ветвления - «условиее прекращения рекурсии»), имеющую две или более альтернативные ветви.

Хотя бы одна является **рекурсивной** (где происходит декомпозиция задачи) и хотя бы одна — **терминальной** (для базы).

Рекурсия

Рекурсивная ветвь выполняется, когда условие прекращения рекурсии ложно, и содержит хотя бы один рекурсивный вызов — прямой или опосредованный вызов функцией самой себя.

Терминальная ветвь выполняется, когда условие прекращения рекурсии истинно; она возвращает некоторое значение, не выполняя рекурсивного вызова.

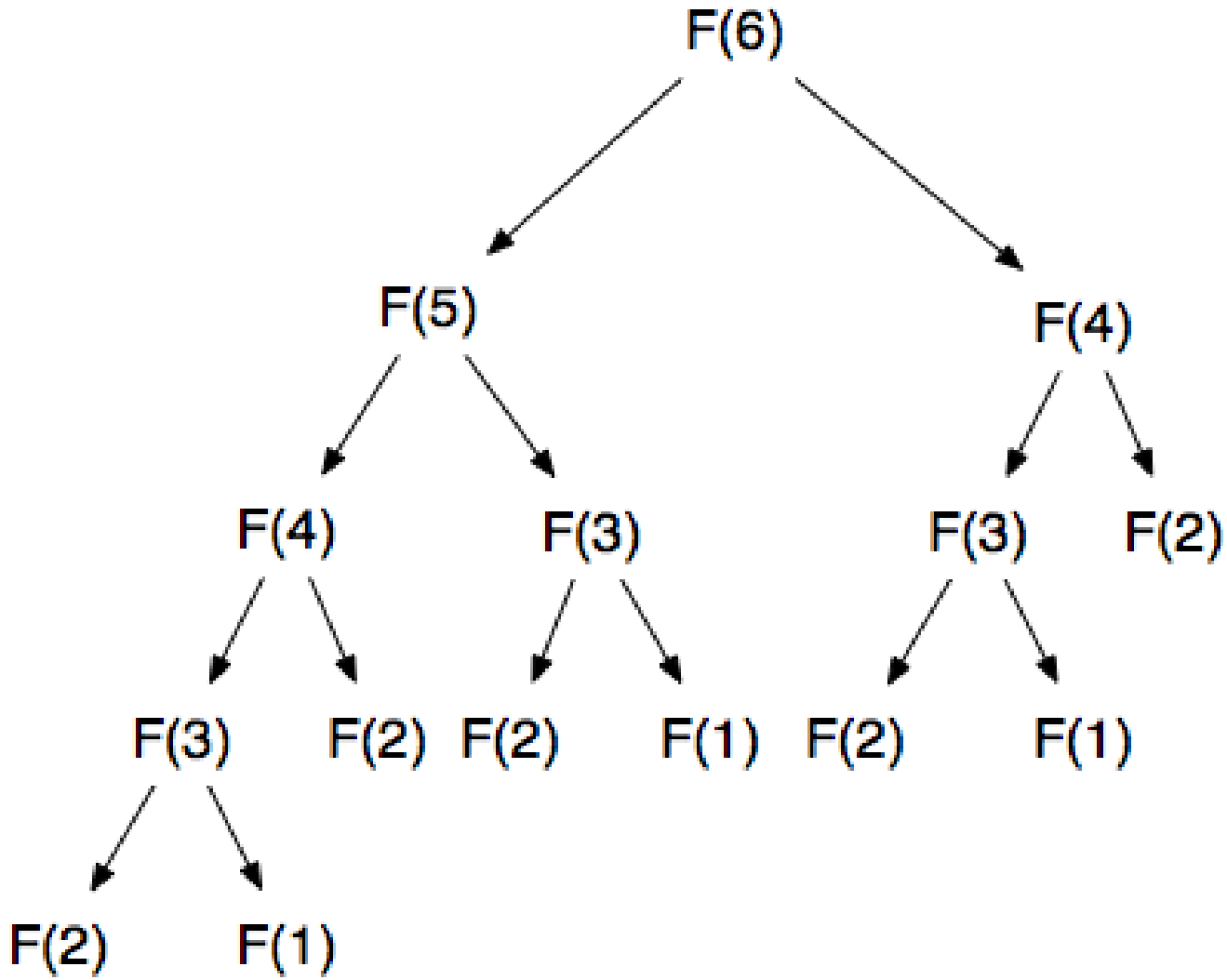
Рекурсия

Пример рекурсивной функции: функция задающая последовательность Фибоначчи

1 1 2 3 5 8 13 ...

$$F = \begin{cases} F(0) = 1; \\ F(1) = 1; \\ F(n) = F(n-1) + F(n-2), \quad n > 1. \end{cases}$$

F(0)=1 - базовая ветвь.
F(1)=1 - базовая ветвь.
F(n)=F(n-1)+F(n-2) - рекурсивная ветвь (декомпозиция).


F(6)
F(5)
F(4)
F(4)
F(3)
F(3)
F(2)
F(3)
F(2)
F(2)
F(1)
F(2)
F(1)
F(2)
F(1)

Рекурсия

- **Простая рекурсия**. Функция F вызывает функцию F из себя собой.
- **Косвенная (взаимная) рекурсия**.Функция F вызывает функцию G, которая вызывает F.
- **Параллельная рекурсия.** Функция F вызывает функцию G, несколько аргументов которой являются функцией F.
- **Хвостовая рекурсия**. Любой рекурсивный вызов является последней операцией перед возвратом из функции

Простая рекурсия

Вычисление факториала

```
def fact(n):
    if n==1:
        # терминальная ветвь
        return 1
    else:
        #  рекурсивная ветвь
        return n * fact(n-1)
```

Простая рекурсия

Вычисление факториала (с использованием тернарного оператора)

```
def fact(n):
    return 1 if n==1 else n * fact(n-1)
```

Механизм работы рекурсивных функций

Прямой ход рекурсии
Рекурсивный вызов подпрограммы оставляет в сегменте стека данные, помещенные туда на предыдущем вызове подпрограммы, и записывает на вершину стека данные текущего вызова.

Обратный ход рекурсии При достижении базы рекурсии рекурсивные вызовы прекращаются и начинается серия завершений вызовов с извлечением из стека сохраненных значений и использованием их для продолжения вычислений. Последним завершается первый вызов.

Проблемы рекурсии

Основа рекурсии - вызов функции. При вызове функции в стек вызовов добавляется один кадр.
Основная проблема рекурсии - исчерпание стека вызовов.
Поэтому не стоит использовать рекурсию там где предполагается множество вызовов.

В Pyhton[4] по умолчанию максимальная глубина рекурсии равна 2000

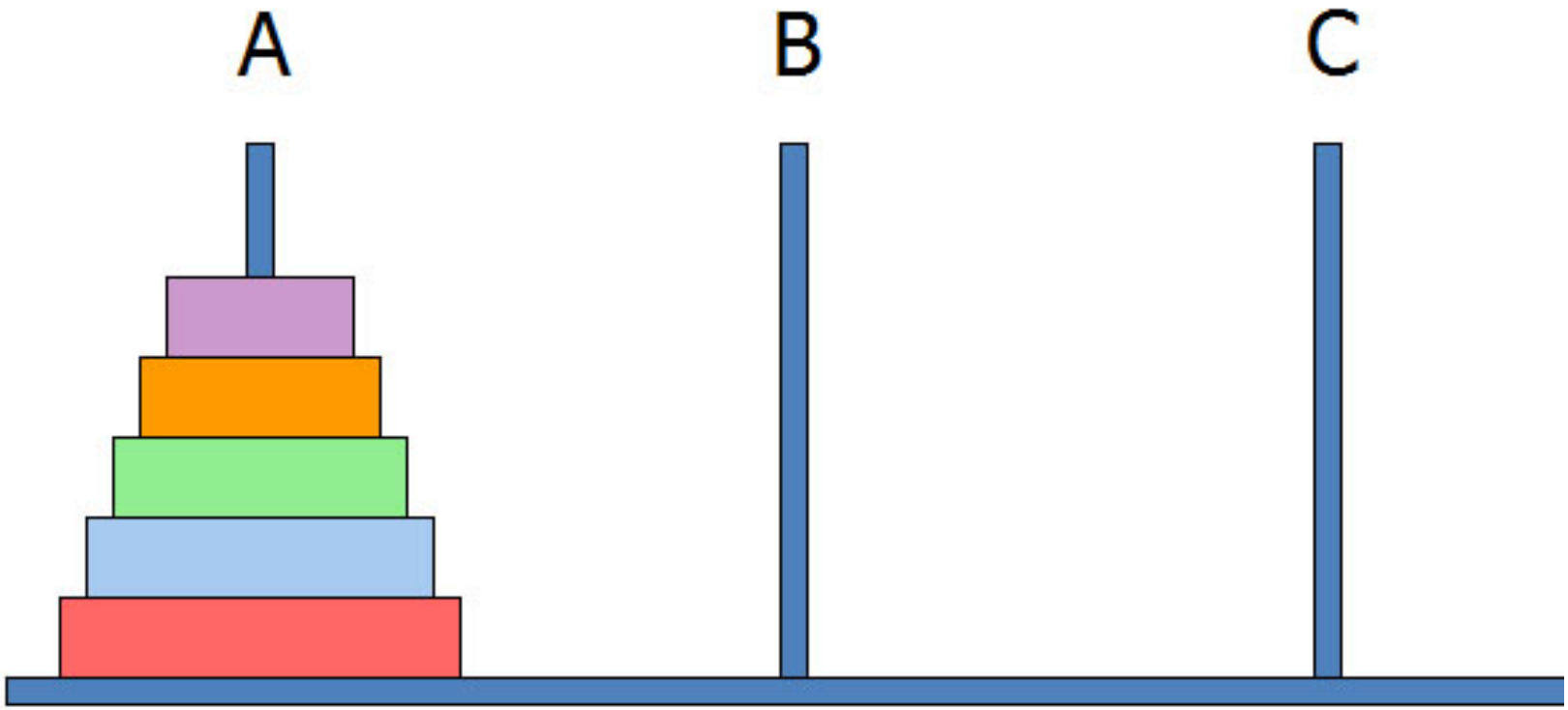
```
import sys
sys.getrecursionlimit()   # 2000
```

[4] Может отличатся в зависимости от версии интерпретатора

Ханойские башни

Даны три стержня, на один из которых нанизаны восемь колец, причём кольца отличаются размером и лежат меньшее на большем. Задача состоит в том, чтобы перенести пирамиду из восьми колец за наименьшее число ходов на другой стержень. За один раз разрешается переносить только одно кольцо, причём нельзя класть большее кольцо на меньшее.

Ханойские башни



Решение

- перенести башню из n1 диска на 2-й штырь
- перенести самый большой диск на 3-й штырь
- перенести башню из n1 диска на 3-й штырь

Решение

- **параметры**: штыри, число перемещённых дисков
- **база**: не осталось дисков для перемещения
- **декомпозиция**:
  - перенести башню из n1 диска на 2-й штырь
  - перенести самый большой диск на 3-й штырь
  - перенести башню из n1 диска на 3-й штырь

Решение

```
# представим диски числами от 1 до n
# 1, 2 и 3-й штыри - списками A, B, C
# требуется перенести диски с A на B
n = 4
A = [ i for i in range(n) ]
B = []
C = []

def move(n, frm, to, buf):
    move(n-1, frm, buf, to)
    to += [ frm.pop() ]
    # pop возвращает последний элемент списка
    # и удаляет его из списка
    move(n-1, buf, to, frm)


move(n, A, C, B)
```

Решение

```
# представим диски числами от 1 до n
# 1, 2 и 3-й штыри - списками A, B, C
# требуется перенести диски с A на B
n = 4
A = [ i for i in range(n) ]
B = []
C = []

def move(n, frm, to, buf):
    move(n-1, frm, buf, to)
    to += [ frm.pop() ]
    # pop возвращает последний элемент списка
    # и удаляет его из списка
    move(n-1, buf, to, frm)


move(n, A, C, B)
```

Где ошибка?

Решение

```
# представим диски числами от 1 до n
# 1, 2 и 3-й штыри - списками A, B, C
# требуется перенести диски с A на B
n = 4
A = [ i for i in range(n) ]
B = []
C = []

def move(n, frm, to, buf):
    move(n-1, frm, buf, to)
    to += [ frm.pop() ]
    # pop возвращает последний элемент списка
    # и удаляет его из списка
    move(n-1, buf, to, frm)


move(n, A, C, B)
```

Где ошибка? Рекурсивная функция не содержит условия выхода!

Решение[5]

```
# представим диски числами от 1 до n
# 1, 2 и 3-й штыри - списками A, B, C
# требуется перенести диски с A на B
n = 4
A = [ i for i in range(n) ]
B = []
C = []

def move(n, frm, to, buf):
    if n > 0:
        move(n-1, frm, buf, to)
        to += [ frm.pop() ]
        # pop возвращает последний элемент списка
        # и удаляет его из списка
        move(n-1, buf, to, frm)

move(n, A, C, B)
```

[5] Для наглядности можно запустить программу в режиме отладки или добавить вывод на экран состояния стержней в функции

Outline

Рекурсия и итерация

Любой алгоритм, реализованный в рекурсивной форме, может быть переписан в итерационном виде и наоборот.

Ссылки и литература

- ИНТУИТ: Рекурсия и рекурсивные алгоритмы
- Викиучебник: рекурсия

Ссылки и литература

Ссылка на слайды

github.com/VetrovSV/Programming